Informatique PCSI
Corrigé TP 6

1 Énoncé des exercices

1.1 Exercice 1

Pour l'affirmation 1 : $8 = 4 + 4$, alors qu'avec un algorithme glouton $8 = 5 + 1 + 1 + 1$.
Pour l'affirmation 2 : $6 = 3 + 3$, alors qu'avec un algorithme glouton $6 = 4 + 1 + 1 + 1$.
Pour l'affirmation 3 : $7 = 4 + 3$, alors qu'avec un algorithme glouton $7 = 5 + 1 + 1$.
C'est l'affirmation 4 qui est exacte.

1.2 Exercice 2

On modifie la fonction `monnaie` données dans le document préparatoire.

```
def monnaie(p, r):
    n = len(p)
    i = n - 1
    solution = {}
    while r > 0:
        while p[i] > r:
            i = i - 1
        val = p[i]
        if val in solution:
            solution[val] = solution[val] + 1
        else:
            solution[val] = 1
        r = r - p[i]
    return solution
```

Test de la fonction :

```
>>> p = (1, 2, 5, 10, 20, 50, 100)
>>> monnaie(p, 48)
{20: 2, 5: 1, 2: 1, 1: 1}
```

Pour rendre 48 euros, on rend 2 billets de 20, une pièce de 5, une pièce de 2 et une pièce de 1.

1.3 Exercice 3

Version récursive de la fonction `monnaie` donnée dans le document préparatoire.

```
def monnaie_rec(p, r, i, sol=None):
    if sol == None:
        sol = len(p) * [0]
    if r == 0:
        return sol
    else:
        if p[i] > r:
            return monnaie_rec(p, r, i-1, sol)
        else:
            sol[i] = sol[i] + 1
            return monnaie_rec(p, r-p[i], i, sol)


p = (1, 2, 5, 10, 20, 50, 100, 200, 500)

print(monnaie_rec(p, 48, len(p)-1))
```

1.4 Exercice 4

1.

```
from random import randint
from math import sqrt

nbpoints, dim = 40, 20
couleurs = ["r", "g", "b"]
```

2. Création des points.

```
def points(n, c):
    pts = []
    while len(pts) < n:
        x, y = randint(-c, c), randint(-c, c)
        cl = couleurs[randint(0, 2)]
        if (x, y, cl) not in pts:
            pts.append((x, y, cl))
    return pts
```

3. Tracer des points.

```
import matplotlib.pyplot as plt

nouveau = (randint(-dim, dim), randint(-dim, dim), "k")
pts = points(nbpoints, dim)
for u in pts:
    plt.plot(u[0], u[1], u[2] + "o")
plt.plot(nouveau[0], nouveau[1], nouveau[2] + "+")
plt.show()
```

4. Modification de la fonction `distance`.

```
def distance(p1, p2):
    x1, y1, c1 = p1
    x2, y2, c2 = p2
    return sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

def distances(pts, dep):
    n = len(pts)
    tab = [(n+1)*[0] for  i in range(n+1)]
    for i in range(n):
        for j in range(i):
            tab[i][j] = distance(pts[i], pts[j])
            tab[j][i] = tab[i][j]
        tab[n][i] = distance(dep, pts[i])
        tab[i][n] = tab[n][i]
    return tab

def indice(position, dist, dispo):
    n = len(dist) - 1
    mini = 3 * dim
    for i in range(n):
        if dispo[i]:
            d = dist[position][i]
            if d < mini:
                mini = d
                ind = i
    return ind
```

5. Fonction `proches_voisins`.

```
def proches_voisins(dist, k):
    n = len(dist) -1
    voisins = []
    dispo = n * [True]
    position = n
    while len(voisins) < k:
        position = indice(position, dist, dispo)
        voisins.append(position)
        dispo[position] = False
    return voisins
```

6. Dessin final.

```
tableau = distances(pts, nouveau)
k = 4
vs = [pts[i] for i in proches_voisins(tableau, k)]
# une liste de compteurs pour compter les couleurs
```

```
cpt = [0, 0, 0]
for v in vs:
    if v[2] == "r":
        cpt[0] = cpt[0] + 1
    elif v[2] == "g":
        cpt[1] = cpt[1] + 1
    else:
        cpt[2] = cpt[2] + 1
i = cpt.index(max(cpt)) # l'indice de la couleur la plus représentée
choix = couleurs[i] # la couleur la plus représentée

for u in pts:
    plt.plot(u[0], u[1], u[2] + "o")

plt.plot(nouveau[0], nouveau[1], choix + "+")
plt.show()
```